Using art services outside art

BNL wire-cell

David Adams

BNL

September 14, 2015

Updated Sep 16, 2015

Introduction

Art and LArSoft provide many services

- TFileService – manage root files and objects
- RandomNumberGenerator – generate randoms, manage seeds
- Geometry – access detector geometry info
- And many more…

Motivation to use services outside art framework

- Convenient to reuse the services outside framework
  - E.g. in Root scripts, user main program for data analysis, event display, …
  - Avoid need to duplicate the effort that went into writing the art services
- User code using services can work inside and outside framework
  - So we don't have to branch for different interfaces depending on context
  - And so avoid ugly cod and extra coding (see following page)
  - Reduce the need for testing in multiple environments
- New user services can be written as art services
  - Easily take advantage of fcl configuration
  - They will also work in art

Ugly code

Like to avoid code like this:

```
#ifdef ARTFW
#include “art/Framework/Services/Optional/TFileService.h”
#include “art/Framework/Services/Registry/Servicehandle.h”
#else
#include “TFile.h”
#endif
.
.
.
#ifdef ARTFW
  art::ServicHandle<TFileService> pfs;
  TH1* ph1 = pfs->make<TH1F>(“hist1”, “My hist”, 50, 0, 100);
#else
  Tfile::Open(“myfile.root”, “CREATE”);
  TH1* ph = new TH1F(“hist1”, “My hist”, 50, 0, 100);
#endif
.
.
.
```

Or introducing another layer which hides such code.

Can we do it?

Art developers were not encouraging

- Art services are intended to be used in the are framework

But also not strongly discouraging

- Didn't say not to try and provided some guidance

So I went ahead and gave it a try

- First goal was to produce a simple main program that configured, loaded and used some services
- Success!
    - I am able to load and use the art TFileService and
    - load the RandomNumberGenerator (didn't try to use it yet) and
    - load and use the geometry service
- Code is a bit ugly and so I hid it in a class ArtServiceHelper
    - Repository: https://github.com/dladams/dune_extensions
    - Code is in DXArt with examples/tests in test/DXArt
    - See some of the code on following pages
    - So far, only supports a single thread

ArtServiceHolder

```
class ArtServiceHelper {

public:

  typedef std::vector<std::string> NameList
  typedef std::map<std::string, std::string> ConfigurationMap;

  // Return the one instance of this (singleton) class.
  static ArtServiceHelper& instance();

  // Delete the one instance of this class.
  // Services are not longer available.
  // The current instance of this class and all services are deleted.
  static void close();

  // Dtor.
  ~ArtServiceHelper() = default;

  // Add a service.
  //   name - Name of the service, e.g. "TFileService"
  //   scfg - Configuration string for the service, e.g. for TFileService:
  //              TFileService: {service_type: "TFileService" fileName: "test.root"}
  // Configuration format is the same as that found in the services block of an fcl file.
  // Returns 0 for success.
  int addService(std::string name, std::string scfg);

  // Load the services, i.e. make them aviailable for use via art::ServiceHandle.
  // Returns the status: 1 for success, 2 for failure.
  int loadServices();
```

ArtServiceHolder (cont)

```
 // Return the names of added services.
 NameList serviceNames() const;

 // Return the configuration string for a service.
 std::string serviceConfiguration(std::string name) const;

 // Return the full configuration string.
 std::string fullServiceConfiguration() const;

 // Return the service status.
 // 0 - not loaded
 // 1 - services loaded and available
 // 2 - service load failed
 // 3 - service helper is closed
 int serviceStatus() const;

 // Display the contents and status of a service helper.
 void print(std::ostream& out =std::cout) const;

private:
...
};
```

test_TFileService.cxx

```
// test_TFileService.cxx
// David Adams
// September 2015
//
// This test demonstrates how to configure and use the art TFileService
// outside the art framework.

#include "art/Framework/Services/Optional/TFileService.h"
#include <string> #include <iostream>
#include "TFile.h"
#include "TH1F.h" #include "art/Framework/Services/Registry/ServiceHandle.h" #include "DXArt/
ArtServiceHelper.h"

using std::string; using std::cout;
using std::endl;
using std::vector;
using std::unique_ptr;
using art::TFileService;
using art::TFileDirectory;

int test_TFileService(string ofilename) {
  const string myname = "test_TFileService: ";
  cout << myname << "Starting test" << endl;
#ifdef NDEBUG
  cout << myname << "NDEBUG must be off." << endl;
  abort();
#endif string line = "-----------------------------";
  cout << line << endl; cout << "Fetch art service helper." << endl;
  ArtServiceHelper& ash = ArtServiceHelper::instance();
```

test_TFileService.cxx

```
cout << line << endl;
cout << myname << "Add and fetch TFileService." << endl;
string scfg = "TFileService: { fileName: \"test.root\" service_type: \"TFileService\"}”;
assert( ash.addService("TFileService", scfg) == 0 );

cout << line << endl;
cout << myname << "Load the services." << endl;
assert( ash.loadServices() == 1 );

cout << line << endl;
cout << "Get TFile service." << endl;
art::ServiceHandle<art::TFileService> pfs;

cout << "Check if TFile is open." << endl;
assert(pfs->file().IsOpen());
cout << "Retrieve TFile name." << endl;
cout << "File name: " << pfs->file().GetName() << endl;

cout << line << endl;
cout << "Add a histogram." << endl;
TH1* ph1 = pfs->make<TH1F>("hist1", "Histogram 1", 100, 0, 100);
.
.
.
```

Split services and geometry

An alternative is "split services"

- I.e. designing service so that the non-art code is in one part and the art interface is provided in a separate piece
- The users outside the art framework can use only the first part and get the desired functionality
- LArSoft Geometry service is already this way—the non-art part is in the base class GeometryCore

My dune_extensions package uses both

- There is an example directly using the geometry service in test_DXArt/test_Geometry.cxx
- There are examples using GeometryCore test_geometry.cxx and draw_geometry.cxx in test/DXGeometry